Applicability of Extreme Programming In Educational Environment

Sultan Alshehri *su.alshehri@mu.edu.sa*
Computer Sciences and Information Technology College
Majmmah Univeristy
Majmaah 11952, Saudi Arabia

Abstract

In order to validate the results of any XP study, the experiment's environment has to be in line with XP principles and values. XP practices must fit with the environment sitting to accomplish accurate finds and observations. This paper searches the applicability of using the Extreme Programming method in the educational environments specifically in the post-secondary school. After digging in 14 XP experiments were done in different universities and institutions, we figured out that XP is applicable in the educational environment with challenges attached. In this paper, we go deeply in investigating the difficulties that students face when adopting XP in class. Also, we highlight the important factors that affect the XP adoption in education.

Keywords: Extreme Programming, One-site Customer, Development Cycle, and One-site Coach.

1. INTRODUCTION

Extreme Programming is one of the most successful methods in the software development. XP offers a set of practices designed to work together and to provide a tangible value to the customer. It brings all the team members including the managers, customers, and developers as one strong node to create a very informative and collaborative team. XP stands on five essential pillars; communication, simplicity, feedback, respect, and courage The software developers promised to deliver a product confidently faster along with maintaining the quality to be very high [1]. XP has received substantial attention in the industry and approved its efficiency and effectiveness in the small and mid size project. Moving towards the education, different expectation will be obtained reflected by the educational constraints and goals.

Almost all the XP practices worked well in the education exclude the 40-hours work per week (inapplicable). Students also found difficulties in applying some of the practices in the context of the curriculums. Lack of experiences, conflict in schedules, availability of the customer, and one site coach have significant impacts on the success of the XP in the educational fields. The controversial practices raise a question whether applying the XP by students will be successful or not? Are the students truly will follow the XP rules or not? How can we adjust the educational sitting in order to help students adopting the XP principles and values?

By reviewing many XP experiments performed by students from different level of education, the answer for these questions will be provided. This paper is organized as follow: in section 2, briefly providing the common purposes of using the XP in education. In section 3, reviewing fourteen XP experiments performed in educational environment .In Section 4, outlining the adoption levels of XP practices. In section 5, we present the comments figured when investigating the XP experiments. In section 6, we discuss the important factors that affect any XP project is done by students. In section 7, we conclude this comprehensive review.

2. PURPOSE OF APPLYING XP IN EDUCATIONAL ENVIRONMENT

There are different goals of using XP in the educational environment. Yet, what do we mean by "educational environment"? In the literature review we targeted students on a high education level for both undergraduate and graduate students studying in computer science or other relevant major that teaches some of the programming languages.

Reviewing more than 17 published white paper, we found the purpose of applying the XP in the educational environment is varied. It can be one of the following purposes:

- Extreme programming or other agile methods might be one of the courses that taught in undergraduate or graduate program and apart of the degree requirement [2,10,11].
- To investigate the effectiveness and efficiency of using the XP practices in the development process [3].
- To Compare the XP approach with other agile methods such as plan-driven development.
- To fulfill project requirements assigned for the fourth undergraduate student or even just a project for a computer or engineering course [8,9,12].
- To find out what are the core problems and difficulties of teaching agile methods in an educational environment [4].
- Evaluation some of the XP practices [4].
- Teaching the student how to collaborate with others as one team, and showing the benefits of working collectively [5].
- Examine some critical factors on XP such as: personality type, learning style, and skill levels and presenting their impacts on the compatibility of pair programming.
- Studying the enjoyment, preferences, the impact of education and experiences.
- Seeking for creative ways that can increase the pair compatibility applied to freshmen, or undergraduate students or graduate student.

3. OVERVIEW OF XP EXPERIMENTS IN EDUCATION

At the University of Karlsurhe, Matthias and Walter [2] experimented some of the XP practices on a graduated course to get a fair evolution of this method in the educational environment. Their study was focused on pair programming, iteration planning, testing, refactoring, and scalability. During only one semester the authors came up with the following observations. First, the students enjoyed the pair programming and 43 % of the participants had learned from their partners. Yet, some of the students did believe that the pair programming is just a waste of time. Second, designing in small increments needs trained developers. Third, due to the lack of the communication among the team members the information exchange was not properly obtained. Fourth, the students found it very difficult to write the test cases before coding. However, 87 % of the students said that the test case strengthened their confidence and improve the quality of the code.

Pair programming experiment was done in the International Islamic University Malaysia [3] to seek the benefits of the technique through Java programming course for under gradate students. The study assumed that all students have similar level of experience of programming. Majority of students (96%) enjoyed pairing with collaborative work and 82 % of the participants enhanced their understanding of coding project and assignments.

Jean Guy and Lorraine evaluated some of the XP practices [4] via students in Software Engineering field, in a learning environment. The study focused on pair programming, metaphor, on-site customer, and development cycle. When investigating the pair programming the authors found that most of the educational systems rely only on the high marks to match the pairs which creates some problems in term balancing the strength of the team members. Also, they found the personality types and genders have a significant impacts on the pair performance. In the development cycle, there is a problem on the short duration spent for working by the teams on the assignments which distinct the educational environment from the industry. In the industry, it is normal to get 40 hours per week working for the project and the iteration can be every two weeks.

On the other side, the maximum time we get in the education is 10-15 hours per week including the regular class and lab sessions. Moreover, the authors highlighted some factors that really affect the XP practices in the academic environment such as the workplace, schedules, customer availability, and the student's level of experience.

Webster University has designed a graduate course to teach Agile Software Development following Object-Oriented programming in the curriculum [5]. The students mainly explored some of the extreme programming aspects such as planning game, unit testing, refactoring, and pair programming. Peter Maher the instructor of the course played a role of the customer and kept observing the students behavior while experimenting the XP process. The students had big problems in estimating the workload and finishing the stories on time. Also, they found that coming with a small design is not easy task. However, in the end of the class, there was a significant improvement observed by the instructor on the whole students' work. One of the important practices aimed by teaching this course is working as a team and sharing the knowledge. The students grasp the benefits of the collaboration and reflected positively that on their project.

A study conducted at North Carolina State University to exam the capability of students pair programmers based on personality type, learning style, skill level, programming self esteem, work ethic, or time management preference. The author found that more than 93% of the pairs are compatible with their partners if they have a similarity in these factors. More than 1350 students participated in this exam from different level of education and experience, fresh man, under graduate students and graduate students [6].

Shippensburg University offers courses in software Engineering that introduced Extreme Programming as a core part of the curriculum in the first two semesters [7]. The instructors require students to complete one project using two different methodologies. In the first semester, the students were required to use a waterfall process for the first release and to use the XP method for the second release in the second semester. The goal of this change is " to show the students extremes of the spectrum of development process". Exercising the two methods students learned that there is no optimal process fits all the situations. The culture of the organization plays a significant role in choosing the appropriate process for the right people. They also believed that using the XP doesn't mean having less disciplined than waterfall. Students showed their skills in the teamwork have incredibly improved. Later on, the instructors made an official change to the software engineering curriculum by substituting the waterfall process by plan-driven development. Similarly, the students had to complete a project using plan – driven and Extreme programming in two semesters. The students were so excited to start the XP more than the plan-driven. In the end of the semester, the instructors came up with three important notes:

1- Switching the partners improves the pairs' performance and helps to spread the knowledge.
2- The students showed negative results in their performance when using automated testing built by one of the faculty members.
3-The students were not successful to design incrementally even though they were taking a class of design patterns and refactoring.

Extreme programming was applied in the 4th year design project at the University of Calgary [8]. The project implemented by a team of four students, three of them had no prior Java experience and only one member had some experience. Indeed, the team members were new to the XP approach. They team could not cover all the XP practices. However, they were able to experiment pair programming, refactoring, incremental deliverables, unit testing, and configuration management. The challenges appeared in the early stage, after four months the team quit the practice of pair programming and felt that it was just a waste of time. Others indicted to the conflict in the pair schedules. The team also had a difficulty in the planning and estimating phase. But because of having a very customer –centric the group produced a functional piece of software that met the customer's desire. The students found the uses cases is very good tools to provide a clear understanding of the expected code and also to display the customer's requirements. In the refactoring process, the students spent very along time to adjust the existing code and found difficulty to integrate the system. Yet, the team experienced the unit testing and ensured its benefits when the risk of errors was decreased.

The students also used the Java Concurrent Versions System as configuration management to reduce the risk of overwriting the code. Dramatically, this tool brought a confidence to the programmers and led the team to release the software.

Shippensburg University offers a 4-credit course that includes all the 12 XP practices [9]. The students had the challenges in five practices, which are pair programming, test-driven, the planning game, the iteration planning game, and refactoring. The students assigned to a project that must be delivered within 14 weeks. Thirty students were participated and divided into two teams of 15 students each. Most of the students worked in pairs in previous courses, so they were aware of the aspect and comfortable with their partners. Nonetheless, the students faced a real problem in arranging extra time to pair outside of the class. Also, they did not switch the pairs regularly which is negatively affected their productivity in the earlier stage. Even though the both teams performed some refactoring, one team enjoyed it more than the other. At the end of the first iteration both teams were failed to deliver functions for the planned stories. The teams were reasoning the failure to the lack of the customer's involvement. This problem was diagnosed and solve by increasing the customer availability, which is finally resulted in bridging the gap of understanding between the customer and the developers.

The most noticeable weakness was in using test driven development especially for the GUI-related class. Caril A. Wellington stated " the test driven development was not natural for our students".

Eventually, the team could develop a successful project with a reasonable code quality and average of 152- production LOC per students. Another XP experience was performed in a software engineering course at the Lund Institute of Technology in Sweden [11]. More than 107 students enrolled to the class and most of them had no priori experience of agile methods. Their programming skills were limited as well. The students had to work in a project during 6-week period. The course was focused in some XP aspects such as planning game, small release, one-site customer. There were some additional aspects such as on-site coach, and team in one room, spike time, reflection, and documentation. Since the students were new to the XP, the instructors insisted to use some of the PhD students to coach the teams. Positively, the coaches kept tracking of the team members and guided the students to apply the XP practices rightly. "Team in on room" aspect brought several benefits to the team in term of increasing communications and having a common vision. Also, it facilitated the job of the coaches to spot any problem easily. Corel Hedin and Lars Bendix promoting the developers to have sometime assigned to work individually "spike time". It helped the team to read and learn about the various practical issues. In addition, the team assigned occasionally a meeting for reflection of the current work aimed to avoid any repetitive problems. One of the new practice done in this course that the students required to have some documentation more than the code only that includes the user stories, the architectural description, and the technical report. The instructors noted that only one team used refactoring tools in the project

Another XP study was conducted in Spring 2004 at the Metropolitan State College in Denver [12]. Eighteen students were participated in a one-semester project. They were divided into three teams of six members each. The Students had limited knowledge about the XP. They were introduced to the XP process through the lecture, textbook discussion, in-class exercise and video session. The instructor played two essential roles, one as customer and second as instructor. He was switching between wearing to colored hats to distinguish the two roles. The students experimented several practices and techniques include pair programming, coding standard, simple design, refactoring, testing, configuration management, and documentation. The students welcomed to be paired and gave a very positive feedback about pair programming in term of writing a great quality of code. However, some students believed that the refactoring is not adding any value to the system. Another point, all the team members have taken their commitments seriously which reflected to the a great collaboration. Here, the instructor found that the small size team was a major factor in having an informative communication. The teams had no problem to release a piece of the software in a small iteration frequently. In the testing phase, the students were also successful with the Junit testing. The major source of the problems was in the "small design" practice, the students struggled to understand the concept behind the simple design. However, The teams generally did well in most of other XP practices. In the in the end of the study, the authors believed that the XP is not suited model for the software Engineering course for some undergraduate students.

In 2001 and 2002, XX taught XP for the undergraduate students in a duration of two semesters and to run three different projects. More than fifty-five students were participated in each project and two professors were involved to play the roles of the customers and coaches. After completing the three projects, XX reached the conclusion of the following points:

1- The amount of the information related to the concepts and practices need enough time to be learned not just in a short semester.
2- Keeping tracks of the students, which were involved in multiple teams is a very difficult task.
3- Some of the XP practices are required advanced programmers or to have some experiences to do it properly. For example, doing the simple design and the writing the test before coding.
4- Having a real customer is not an option in most of the educational environment. Yet, acting as a customer required a willingness to be always available, which is also difficult for whoever is acting as a customer.
5- There is a challenge to ensure that if the students are focused on high quality and well-factored code!
6- Extreme programming cannot be taught in one semester.

Grigori and Frank conducted a XP study in Southern Alberta Institute of Technology [13]. The experience involved forty-five students enrolled in the three different academic programs (Diploma, Bachelor's, and Master's). The authors determined the following perceptions: (1) the students generally were very enthusiastic about the extreme programming. (2) The majority liked at most the "planning game" and noticed a great improvement in the accuracy of their estimation. (3) In the pair programming, some pairs faced a real problem caused by their weak and unskilled partners, so a suggestion was brought by students to match the two partners based on the qualifications and experiences. Other problem regarding the pairing was caused by the intervening between the project schedule and other outside commitments the students had.

(6) The Test- first design was very confusing for most of the students. They believed that, thinking the test first is like working backwards.
(7) The most surprising result was when the authors stated at the end that "there were no significant differences in the perceptions in the perceptions of the students of the various levels of educational programs and experience".

A special study in pair programming conducted by NSF at the University of California indicated that the sources of difficulties in pairing were because of the disparity in the experiences, scheduling conflicts, lack of understanding and unbalance of driving and reviewing time [14].

The authors also found misconducts accrued by some students when submitting the assignments with both partner's names, even if one of the partners has done nothing. Investigated showed that those who cheated were focused on getting good marks more than focusing to do pair programming. Also, they found that the time pressure forced students to partition the problems trying to solve it separately, ignoring benefits behind pair-programming concept.

After 10 weeks observation, the instructors highlighted the skills, the experiences, and the cultures to be the most important factors should be taken into the consideration when pairing.

Another study was conducted at the same institution to evaluate main contributors in increasing the compatibility of student pair programming. This study involved more the 361 undergraduate and graduate software engineering students. After three academic semesters, the research showed that the pairs are more compatible if student with:

“Different personality types are grouped together, similar actual skill level are grouped together, similar perceived skill level group together, similar self-esteem are group together, same gender group together, similar ethnicity are group together”.

Positive results were received from the students in Dundalk Institute of Technology that exercised the XP over two months [15]. The most activities were focused on the core development practices: simple design, pair programming, testing, code standard, collective ownership. Students enjoyed the experience of pair programming. They showed 15% of the defect was reduced. More than half of the students prefer to be paired with a stronger programmer. 32 students completed their test cases while the failure of the first exercise ranged from 0%-60% which is indicted some challenges students had to perform this practice. All the students responded that they had applied the code standard. One of the good feedback about the study when 37 students felt that the code belonged to both members.

Forth year students in university of Sheffield have applied the XP concept on a real system for a real customer [16]. The students had very good experience in programming, but none of them hared about the XP before. Each student required working 10-12 hours per a week for the project. 80 students were participated and 3 business clients were involved from Genesys. The study focused on how to communicate with the client and capture the real requirements to deliver high quality of software. The students dealt with two projects, existing one that needs some major testing and debugging and a new project started from a scratch. The authors found out the students can carry real empirical experiments if the relation with customer was fully built.

4. ADOPTION LEVEL OF XP PRACTICES

Most of projects reviewed in this paper attempted to adopt the whole XP practices, but not all of them succeed to achieve this goal. Educational conditions such as the number of students, the project duration, and the level of student’s experience draw the boundary of the possible practices can be applied. These conditions will be discussed as follow:

1- Number of participated Students (Team Size): Team size is a main factor in the communication principle. A large team can negatively affect the communication and create more gab among the team members. On other hand, The small team will allow to collaborate effectively and to get a feedback quickly. Most of XP proponents strongly believe that smaller teams are more functioning and productive comparing to larger teams. However, defiling the optimal number is a challenge. Jeff Sutherland says “There is plenty of data to show that team sizes over 7 result in significantly lower productivity” [23]. While Jurgen Appelo suggested that five is the optimal team size [24]. Several researches and experiences showed that five is the most preferable number to form a productive team but there is no consensus. Other important factors can play roles in the team are: the number of tasks, the communication channels and the expectation of the team’s performance. According to PMP the Communication Channels can be calculated by using the following formula: [N (N-1)]/2 where N equals the number of people involved [25].

2- Project Duration: The available time is a significant constraint in the educational environment. It directly affects the team relations and the whole project process. It is not easy to do several missions in one academic semester such as teaching the XP, training the students to follow the process, and applying the XP practices only. However, most of the XP experiments presented in this paper showed satisfied results during this short period.

3- Level of Experience: the students are categorized in one of the three educational levels: undergraduate, diploma, and graduate levels. The programming knowledge and solving problems skills are part of the experiences, but the real work involvement is more valuable than anything else.

In table1, we compare the 14 experiments reviewed in this paper from four perspectives: adopted practices, number of students participated, education level, and project duration.

	Applied XP practices	**Participants**	**Experiences Level**	**Duration**
1	(1) Pair Programming (2) Iteration planning (3) Refactoring (4) Testing (5) Scalability.	12 Students (2 teams)	CS graduate Students had experience with the teamwork.	8 weeks.
2	(1) Pair Programming.	34 Students	-All students have similar levels of the experience. - Undergraduate Students.	One Academic Semester.
3	(1) Pair Programming (2) Testing	Not Specified	-Undergraduate Students.	2 Academic Semesters.
4	(1) Pair Programming (2) Refactoring (3) simple Design (4) Incremental Deliverable (5) Use Case Requirement (6) Unit Testing (7) Configuration Management.	4 Students	- Undergraduate Students. -No experiences in Pairing.	4 Months.
5	(1) Planning Process (2) Small Release (3) Pair programming (4) Metaphor (5) Simple design (6) Coding Standard (7) Continues Integration (8)Collective code Ownership (9) Refactoring (10) Testing (11) One-Site Customer.	15-16 Students (Two teams).	- Undergraduate Students.	14 weeks
6	(1) Pair Programming (2) Whole Team- Testing (3) Code Standard (4) Simple Design.	18 Students (Three teams)	- Limited prior knowledge of XP.	One Semester
7	(1) Unit Testing (2) Refactoring (3) Pair Programming (4) Metaphor (5) One-site Customer.	Not specified	-Graduate Students.	One semester
8	(1) Pair Programming (2) Planning game (4) Test-First Design (5) Collective code ownership (6) Continuous integration.	45 Students	-Various level of the experience starting from second year up to graduate students.	12 weeks.
9	(1) Pair Programming.	Not Specified	- Undergraduate students (freshman with no experience)	2 semesters 14 weeks.
10	(1) Pair Programming.	316 Students	-Undergraduate Students. -Graduate Students.	3 Academic semesters.
11	(1) Planning Game (2) Small Release (3) On- Site Customer Additional practices: (4) On-Site Coach (5) Team in One Room (6) Spike Time (7) Documentation.	107 Students	- Undergraduate Students. - Most of them had no experiences.	Seven weeks
12	(1) Planning Game. (2) Short release (3) Testing (Unit acceptance) (4) Refactoring (5) Pair Programming. (6) Collective work (7) Continuous integration (8) Coding Standard.	1: 55 Students 2: Specified. 3: 70 Students	-Undergraduate Students.	1:8 weeks. 2: 12 weeks. 3: 6 weeks.

13	(1) Planning Process (2) Small Release (3) Pair programming (4) Metaphor (5) Simple design (6) Coding Standard (7) Continues Integration (8)Collective code Ownership (9) Refactoring (10) Testing (11) One-Site Customer.	80 Students (5-6 teams)	-Undergraduate Students	12 Weeks
14	(1) Simple Design (2) pair programming (3) Code Standard (4) Refactoring (5) Testing (6) Collective Ownership.	57 Students	-Diploma Software Development	2 Months.

TABLE1: Comparison Among the Extreme Programming Experiments.

U: University as follow: (1) University of Karlsurhe (2) International Islamic University Malaysia (3) Shippensburg University, (2000) (4) University of Calgary (5) Shippensburg University (2005) (6) Metropolitan State College of Denver (7) Webster University (8) South Alberta Institute of Technology (9) University of California at Santa Cruz
(10) North Carolina State University (11) Lund Institute of Technology Sweden (12) University of Auckland (13) University of Sheffield (14) Dundalk Institute of Technology.

5. COMMENTS ON THE XP'S 12 PRACTICES APPLIED BY STUDENTS

This section briefly discusses the 12 main practices of the XP and how they are benefiting the software development as whole. The challenges faced by students will be presented as well. The XP practices are: the planning process, small release, metaphor, simple design, testing, refactoring, pair programming, collective worship, continuous integration, 40-hours work week, on-site customer, and coding standards (Beck, 1999).

5.1 Planning Process

Planning release is a sharing effort between the customer and the programmer. In the beginning of the project, the team will hold a release planning session to identify the features of the system. The stories are the most important valuable feature in this session. The word of " story" is replaced the requirement in the waterfall method. It can be seen as the unit of the functions in XP [17]. In the educational environment the customer is needed to participate in the release planning. It might a problem to have a real customer from outside. However, in most cases, the instructor plays the role of the customer. But he still needs to act as instructor for the education purposes. If action of switching roles between the customer and the instructor did not go well, it will mislead the student and create obstacles in applying the XP process.

The role of the customer in the planning process is defining the user stories, determining the business value for the desired feature, and prioritizing the stories that will be implemented for the release.

On the other hand, the programmers (students) estimate each story (cost and time) and present the potential technical risks to be avoided. Both customer and programmer must establish the schedule for the release and the duration for the iteration. They answer two questions: what should we work first? What will be working on later? The students need some help form the instructor to overcome the following difficulties [5,10,13]:

(1) Breaking the big stories into smaller stories.
(2) Knowing how many stories they will be able to do per iteration.
(3) Estimation how long each story will take.
(4) Responding to the changes when the customer changes his mind about the requirements or his priority about the stories.
(5) Measure the team progress " Velocity".

Since this practice acquires some experience, the instructor has to provide roadmap for students to reach their final destination. The instructor also has to switch his role to play the customer's role and participate in forming stories, planning game, iteration planning, and estimating.

5.2 Small Release

The XP team should release iterative version of the software after is being tested. The release rang from a day to a month. The customer excepting to see the features he agreed to be done in the iteration.

[release frequently does not mean setting aggressive deadline].[in fact, aggressive deadline extend schedule rather than reducing them[McConell 1996,p.220]. The idea is to minimize the functions and maximize the confidence and trust. In other word, it is better to release a small set of functions that works properly and bring tangible value to the customer rather than having more features with its associated problems and losing the customer trust.

After the customer sees the release, the feedback and comments can be given to the XP team before going to the next iteration. The customer can cancel the work or change the priority order for the stories. Also, He might see unimportant features that can be removed and ask for the features he finds most valuable. The most difficult part in this exercise found by students is how to make small release regularly. However, they found real benefits can be summarized as follow [9,10]:

(1) Increase the trust and confidence between the customer and the programmers.
(2) More feedback and comments will help to improve the software.
(3) Reduce the risk and pain that team might have through the lengthy cod and testing.
(4) Reduce the pressure on the team when more stress carried on the deadline.
(5) Improving the estimating skills and speed up the velocity.
(6) Keeping testing the software reduces the defect rate.
(7) Ease the step of make simple design.

5.3 Metaphor

Explaining the system to somebody has different perspectives can be a headache and repetitive task. The solution provided by the XP method is " metaphor". It is possibly to meet a person saying "Explain it to me like I'm a four- year-old" [18].

So, the metaphor will help to create a common vision about the system between the customer and the developers/programmers. It is very powerful technique and needed to bridge the different backgrounds and understanding. Moreover, the developer themselves will better their understanding about the system by giving examples and relating it to another area. On the other hand, the customer will be comfortably discuses the system and add beneficial comments about the functions.

In the educational environment the instructor who plays the role of the customer in reality has more experience and understanding than students (not always) about the project and the needed steps. The metaphor is needed by students not buy the customer which is a bit wired and unusual. When the instructor acts as a customer, he must be careful not to act as a developer or coach otherwise this practice " metaphor " will fail to accomplish its job. The " metaphor" is the translator between two languages, programmer language and customer language! When the customer became programmer no need for " metaphor" which possibly occurs in the educational environment.

5.4 Simple Design

The idea in this practice is: " design for today's code with no extra design for tomorrow's work". They system in XP is designed gradually. There is no need to support the next iteration when you work on features for the current iteration. Design enough for something can be tested and running. A simple design is important to avoid the consequences that brought by complex design. For example, when a project has a very detailed design, it will need more testing, more refactoring, more time to fix bugs and defects, more feedback, and conflict understanding. (perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away).

When students trying to apply the idea of a simple design the find a challenge to answer two questions:

(1) What is the simplest design we can come up with that could work?
(2) Is the design ready to be modified easily any time?

From studies and experiences simple design should have the following characteristics [19]:

1- Code and test can be communicating with everything you want.
2- No duplicated code.
3- Have fewest possible classes.
4- Have the fewest possible methods.

If the instructor could not understand the students' design that means they are far away from the concept of the simple design. The simple design is easy to modify and maintain. Yet, there are some difficulties brought by students such as [2,11]:

(1) Students mix the two concepts " simple design " and " simplistic design". Some of them provide a poor design and claim it to be a simple design.
(2) They cannot distinguish between the design for today and design for tomorrow.
(3) It is very hard to avoid the duplication in code.
(4) It is difficult to think ahead about the possible change when designing now [2]. Simple design requires continuous refactoring.

Overall, some students find the design is not very sophisticated task, and some others still think of it as a complicated job.

5.5 Collective Code Ownership
Students liked the idea of " nobody has authority more than anyone else regarding any component of the system". Which is means all the students as one team share the same responsibility. Anyone can work on any part of the system anytime. There are no methods or classes owners. No permission required doing some changes on the code. However, the students found that the collective work can be a problem when changing made by student has less understanding of the written code.

5.6 Refactoring
Refactoring is the process of changing the design of your code without changing its behavior. The developer has to finish all unit and integration test before refactoring. In addition, doing some change to the existing code needs a courage and confidence that can be obtained by complete testing.

The students found the idea of "codesmells" very helpful to answer the common question "When do you need to refractor?" "kent back came up with the idea of code smells [fowler1999]. It means you have kind of feeling that your code smells bad or your peer finds a difficulty to understand the code. It does not mean always the code is bad. However, it can visually determine by the programmer himself. There are many indications that can tell the code smells bad. Here is summarized [20].

Students found the following results of the refactoring practice [8]:

(1) Enhances the design of the system.
(2) Cleans the code and increase the readability.
(3) Minimizes the code debt.
(4) Eases the process of adding a new function.
(5) Finds the bugs easily.
(6) Makes the comments more meaningful.

Matthias and Walter refer the lack of refactoring to two factors: small size of the project and doing complete design rather than simple design [2].

6.7 40-Hours Work Week
Generally, people would need to work overtime because they have to deliver what they promised to be done on the due date. As result, the quality and productivity will be dropped dramatically.

Asking more time is an induction of a big problem including poor planning, wrong estimation, and overcommitted deliverables. XP encourages the team development to stick on 40 hours per week or less. People are happy to go home on time with enough progress for the day as estimated before in the planning phase. Using this strategy improves the team's skills and has positive impacts on other XP practices.

This practice is impossible to apply it in the educational environment. Usually, students have XP course beside other courses during the academic semester. If we assume that the XP class takes 4 hours per a week as regular lectures and labs and it takes 6 extra hours, that means the maximum we can get is 10-15 hours per a week [4]. Keeping in mind the students are not willing to spend too much time for one class and ignoring other classes. Fairly enough if the really students committed to spend 10 hours for the XP including the outside meeting and lab work. As result, this time constraint must be taken into the consideration when:

(1) Assigning the project to the students.
(2) Providing the communication channels.
(3) Sitting up the lab hours.

The project than can be implemented by expert people using 40 hours per a week should be different from a project that implemented by students having 10 hours per a week. The instructor can request the students to keep touch and working with the other team members even from home using Skype or wiki or other tools.

5.8 One- Site Customer
One of the most essential requirements in XP is to have the customer available all the time to sit with the team, answer the questions, and resolve disputes. Customer takes a decision that might affect the business goals. The communication with the customer, face to face in all the XP phases is highly recommended. Moreover, the customer has to take the commitment of his presence seriously to avoid any conflict in the feature between the developers and the business representatives. The XP customer is a part of the team and involves in most of the development process not just a normal customer [10].

In the educational environment, it is difficult to bring a real customer from outside (not impossible), so the instructor usually plays the role of the customer [11].

The availability of the customer (instructor) outside the regular class is an issue, but it can be resolved. If the instructor is not willing to put some extra time to meet with team, he should assign other alternative from the beginning of the project. Otherwise, the failure will be a normal result for this action.

The students will need someone to act as customer for the following reasons [11]:

(1) To make the decisions based on business goals.
(2) Discuses the stories that will be included in each iteration.
(3) Agree about the time scheduled for the release.
(4) To write and prioritize the stories with developer assistance.
(5) To tell the developers how he wants the system to be functioning.

Also, in the programming phase the customer is needed for the following reasons:

(1) To provide the programmer detailed thoughts about the feature and the system behavior.
(2) To write the acceptance test before releasing the system.
(3) To put their feedback and comments after the system features have been tested.
(4) After reviewing the system the customer will give the green light to the developer to continue producing.

5.9 Pair Programming
Students can achieve numerous of benefits when using the pair programming such as:

(1) Pairs learn from each other, (2) less defects, (3) design quality, (4) problem solving, (5) pair – pressure ensures timely delivery, (6) less distraction, (7) higher productivity, (8) satisfaction, (9) team building and communication, (10) storing code ownership, (11) sharing knowledge amongst team members, (12) support self discipline, (13) provides the feedback more quickly than schedule inspection (14) Many mistakes get caught, (15) the end defect content is statically lower, (16) designs are better and code length shorter, (17) students learn more about the system, (18) the project ends up with multiple people understanding each piece of the system,
(19) the people earn to work together and talk more often together. [2,3,5,6,8,14]

However, Most of the studies showed there are common mistakes done by the pairs such as:(1) Absence of one of the pairs during the work time (2) Only one person does the job (3) The pairs are not switching.

The first thoughts and responses by students towards the concept of the pair programming can be one of the following [2,6]:

- Pair programming is a waste of time. They wonder how the team will benefit from the idea of having a person sitting silent beside somebody is coding.
- Pair programming is a waste of effort. They think one job (programming) can be done by one person not two people. The other person can be beneficial for another task.
- Pair programming is not comfortable. Some students have their own style of coding and their way of handling the bugs, so they are not welcoming to collaborate with another person who has different styles and thought.
- Majority of students never paired. However, even after pairing there is portion of students still prefer coding on their own.
- Some of the students feel they are forced to pair but they are not convinced.
- The seniors think the work will be done slowly due to the juniors.

Therefore, The instructors have to prepare answers for the for the challenging questions below:

(1) How to convince students who have no idea about the XP to be paired and to believe of the efficiency?
(2) How to structure the pair! How to choose two students to be paired together? How to make sure that the two pairs are matched? (Primary grades)
(3) How to eliminate the impacts of the personal characteristics conflicts between the pair?
(4) How to evaluate the work properly?
(5) How to overcome the schedule conflicts. Do they students need to pair all the time? Can the pair be remotely?

From other perspective, it is very important to focus in two main issues that affect the success of the pair programming [21]. These two issues are (1) matching between the pairs (2) sitting the workplace for the practice.

(A) Matching the pairs:
It is an essential step that can be performed by the instructors or the students themselves. Whoever takes the responsibility of matching the pairs, has to be aware of the following Factors:

(1) Personality types. (2) Experience. (3) Programming skills. (4) Availability for pairing. (5) Communication channel. (6) Gender. (7) Culture. (8) Language. (9) Appearance.

(B) Environment Sitting:
Most of the labs in the education system are designed to for students to work in individually not in pair. So, it is a matter to sit up the pairing station, which is plays a significant role in XP generally, and pair programming specifically. Most of the labs in the universities have a straight desk, which is not comfortable for doing pair programming. So, the most proper workplace for pair programming should have:

(1) Computer Desk: The two people should sit beside each other comfortably; means the computer desk must be customized for this purpose. Both developers get a straight angle to the screen. The flat desk will be preferable. Also, it is advisable to avoid any desk curve toward a person, which will make the working, condition for paring very difficult.

(2) Screen, keyboard, and mice: It is good idea to have two screens mirroring the display combined with two keyboards and mouse. Each one feels like he has his own place work. However, some developers prefer to work on one of each (screen, keyboard, and mouse).

5.10 Testing
Testing is about finding problems and presenting all the stories were not implemented. The programmer and customer share the responsibility to get this practice done together. Test- driven development [" it is a rapid cycle of testing, coding, and refactoring. When adding a feature, a pair may perform dozens of these cycles, implementing and refining the software in baby steps until there is nothing left to add and nothing left to take away] [book: the art of agile development]. TDD is the heart of XP's programming practices and helps the programmer to make sure that their coding acts exactly as they think. It is about producing small increments to increase the code quality and avoid the massive list of outstanding bugs. Yet, some studies showed that the students spend 50% of the project time for testing [17]. Other studies showed even more, in Dundalk institute of Technology 95% rated time spent developing test cases as productive [15].

The testing normally follows the process of: (1) Creating a test for the small feature. (2) Writing a simple code that makes the test pass. (3) Repeatedly, creating a test for another feature and write the very simple code. (4) Continuing until no story (problem) left to be tested. Moreover, to accomplish a great testing, the tester must avoid duplication, have a good design, choose right names, and keep refactoring. Also, testers have to provide the tests for all possible uses for the software.

Different types of testing will be part of the XP testing phase as follow [8]:

(1) Customer test (called customer acceptance test)
The customer writes the acceptance test a story by story. It is created from the user story and each story can have more than customer test. But don't consider the user story as full unit test. Therefore, the user story cannot complete without acceptance test. The customer will be in charge of reviewing and verifying the correctness of the test. The programmer makes sure the customer requirements are met and the system features are accepted. The benefits of this practice are to bring the customer ideas and thoughts to the team. In addition, He has to agree about the system behavior. Acceptance test also can be used as a measure for the development progress [7].

(2) Integration test
Any test requires further setting for your current system such as communicating with Database or doing tasks across a network is Integration test. When students need to do more integration test, it is a sign of having poor design.

(3) Unit test
It is about testing the class and methods. It helps students to consider what needs to be done. The unit test must go 100% right with no single error. If there is an error, it must be fixed before adding

another feature to be tested. So every single line of code must run correctly [22].

Here are the most notable observation found by applying this practice [2,7,8,9,10,11]:

(1) Experienced testers are needed to work with the customer. A professional tester could make a great contribution to the team. In the educational environment, might have the customer (instructors) have more experience than the programmers and testers (students).
(2) The GUI test is very difficult and complex since most of the GUI was not designed for testing in mind. One of the solutions suggested by Carol A. Wellington, is to make the GUI code as thin as possible and to provide an alternative input mechanism for testing the user interface.
(3) The automating unit test can be very beneficial is some situation. But it can be more costly than the manual test. The problem with automated as Brain Marick said [what's hard about the automated testing is that the product changes, which tends to break the test. If you spend all your time fixing broken tests, was it really worthwhile to automate them?]" [Marick, 1998].
(4) The team will have a big problem if the customer is not be willing to be a core part of this task which is in somehow indicates clearly the broken relation with the customer. However, if the customer cannot write the functional test because of his poor skills, further help should be available. The customer also in this matter should give the examples and the programmers share some possible seniors and get the comments from the customer and apply it.
(5) One of the common mistakes is that the programmers and testers do all the work for the customer testing. It was said "If you know the code (programmer), it is better not to test it alone. Because you will know how it works".
(6) Students were surprised how small each increment can be. This technique is not indication of lack of experiences. On the contrary experienced programmer make smaller increment than the freshman.
(7) Testing task including the acceptance test is a collective effort taken by entire the team members.
(8) Producing excellent tests is result of combination of working between experienced tester and the customer when the programmer takes the role of playing a tester.
(9) Good feedback will bring good testing.
(10) The design will be influenced by the testing and improved until it meets the customer's desire.
(11) Unit test and integration test bring the confidence to the developer making changes anytime without fear.
(12) Many students found the writing the test before the code is extremely difficult [1].

5.11 Coding Standard
The only way to let the students understand the code of each other is to standardize their code. Students in North Carolina State University were instructed to use the Sun Java coding standard. The feedback was very positive 94
% liked it and practiced it [22].

5.12 Continues Integration
In the traditional waterfall process, the integration is a hassle task that can consume a lot of time just to reveal the design and code deficiencies. However, Integration with the XP is ongoing task that keeps the software always verified and tested. Students, found this practice very helpful to catch the problems and fix it at earlier stage. Students are encouraged to have the on site coach to guide them, how possible to integrate the system continually.

6. INFLUENTIAL FACTORS ON THE XP EXPERIMENTS IN EDUCATION
There are many Influential factors that have direct impacts when applying the XP by students at any post secondary schools. In this section, we will discuss in more details these factors as follow:

6.1 XP Team

Extremeprogrammingteamineducationwillincludethefollowingpeople:

1- Students:
The team in the education is different from the team in industry. One of these differences that in the education, we have no options to exclude any of these students registered in the XP class. All students will be part of the project. Students play the role of the developers (programmers and testers). The students must have a minimum of programming skills in order to work part of a productive team. Most of the XP experiments appliedintocomputerscienceorsoftwareEngineering students. Also, the XP exercise should not apply to students who are in the beginning of their study (first and second years). The picture of the software development should be pre-taken in some courses before getting the students into an advanced topic in software process. The students schedulewillmakesignificantimpactsonthewholeprojectsinceextratimeis extremely needed.

2- Instructor:
Different responsibilities carried by the instructor, teaching is the highest priority. XP Teaching process will take various forms of lectures, labs, books and papers ect. Above of that, he will be supervising the whole students' work for grading and educational purposes. Motivation and encouragements giving by the instructor will pass an important energy to the students. Instructor must provide the students with a quick communication way to be reached faster for any questions. Another essential task related to the instructor's mission is scaling the project, watching information exchange among the students.

3- Customer:
Who is the customer in the educational environment? Sometimes instructor takes the role of the customer. That means a heavy load will be carried by the instructor, one as a teacher another as customer. Switching between the two missions can mislead the students. To have a real customer will be the best choose. Any customer will be part of the team has to be fully involved in the whole development process. Also, the relation and trust with the other team members should be strong enoughtogiveappropriatefeedbackandcomments.

4- Coach
Students had no experience need to be trained how to apply the XP practices and to follow its process properly. One- site Coach will benefit the students in this matter, at least in the earlier stages of the project. See figure (1) that summarizes the team members in XP in education

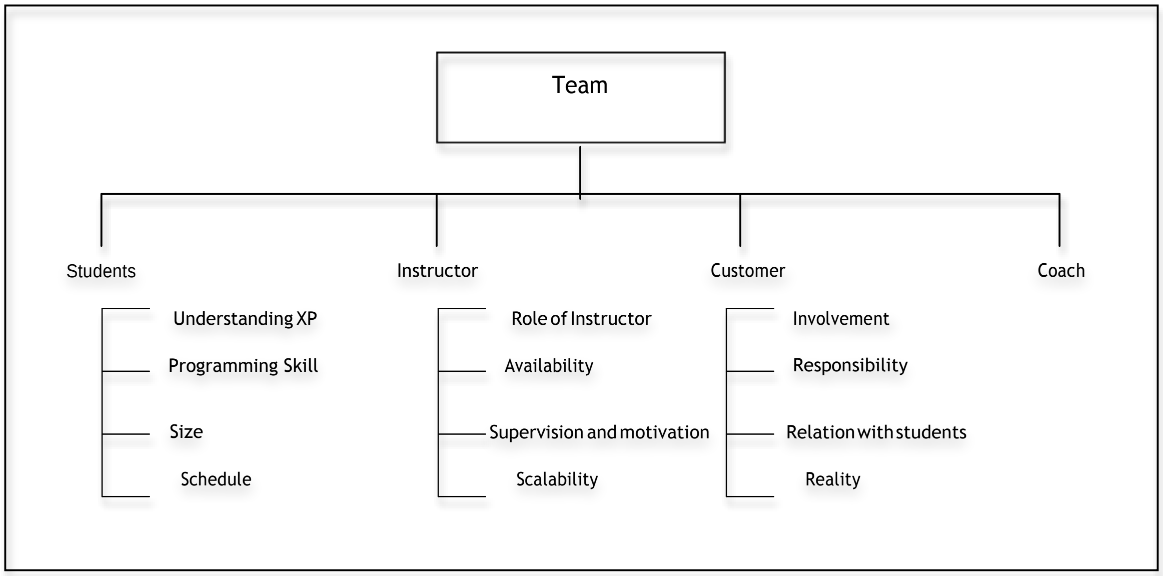

FIGURE (1): XP Team

6.2 Education Objective

An instructor in educational environment has a task of teaching students whatever in the curriculum and then grad the students based on their performance on the class. When adopting the XP experiment on students, the instructor will face a challenge to balance between the education goals and his involvement to work among the team members. Instructor has to present the XP concept, observe students, answer the questions, and work with team, so switching between these roles might deviate the education aims. Moreover, XP encourages by its practices (ex. pair programming, code standard, and code ownership) to work as one team and everyone is responsible for the success in the project. So, evaluating the students' performance on such project done collectively is very hard task.

On the other side, the students want to learn, but getting high marks will also affect the truth of following the XP process. Studies showed in pair programmer as example, some students let their pairs do their jobs and they submit the assignment as if they have done it together which is not true. Students might focus in passing the class more than following the XP rightly. Therefore, the results observed in the end of XP experiment might not be accurate. See figure (2).

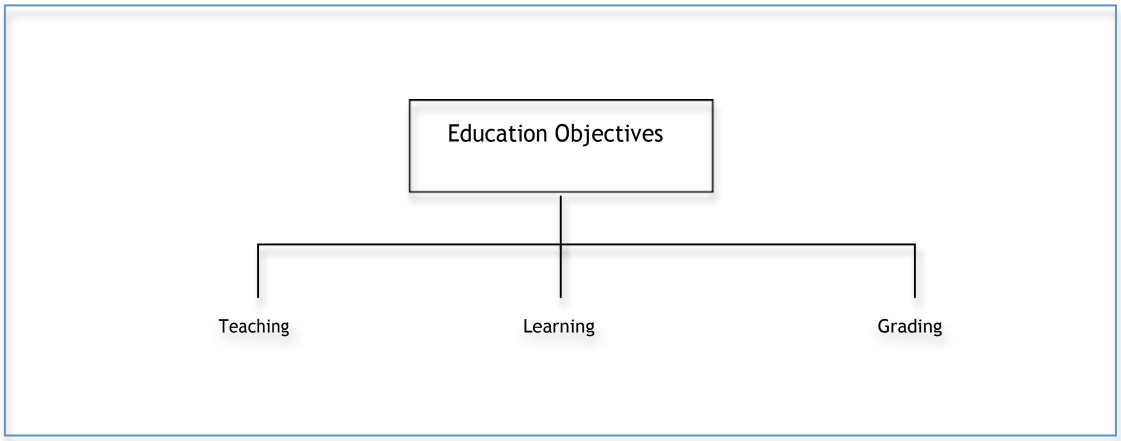

FIGURE (2): Education Objectives

6.3 XP Practice Issues

Investigating the XP practice and its cycle by students with educational constraints brought three important issues:

(1) Missing Practice: 40-houres per week is the only practice will be impossible to do in the educational environment. However, using the other communication tools can make up extra time to be added to the project.

(2) Difficult Practice: the research shows that the most difficult practices students had are: simple design, pair programming, refactoring, and testing. They need more efforts to be paid by all the team members including the instructor and the coach to help the students overcome the obstacles involved in these practices.

(3) Development Cycle: XP is iterative production. The team should release a piece of working software in each iteration, which is quite different from the traditional process that students are aware about. The team will possibly fail in the initial iterations, but things will slowly progress. If the students could not overcome the estimations and planning issues, they may keep failing to deliver any piece of the software on time. The XP principles need to be understood by students to motivate applying the XP practices. XP issues classification can bee seen in figure (3).

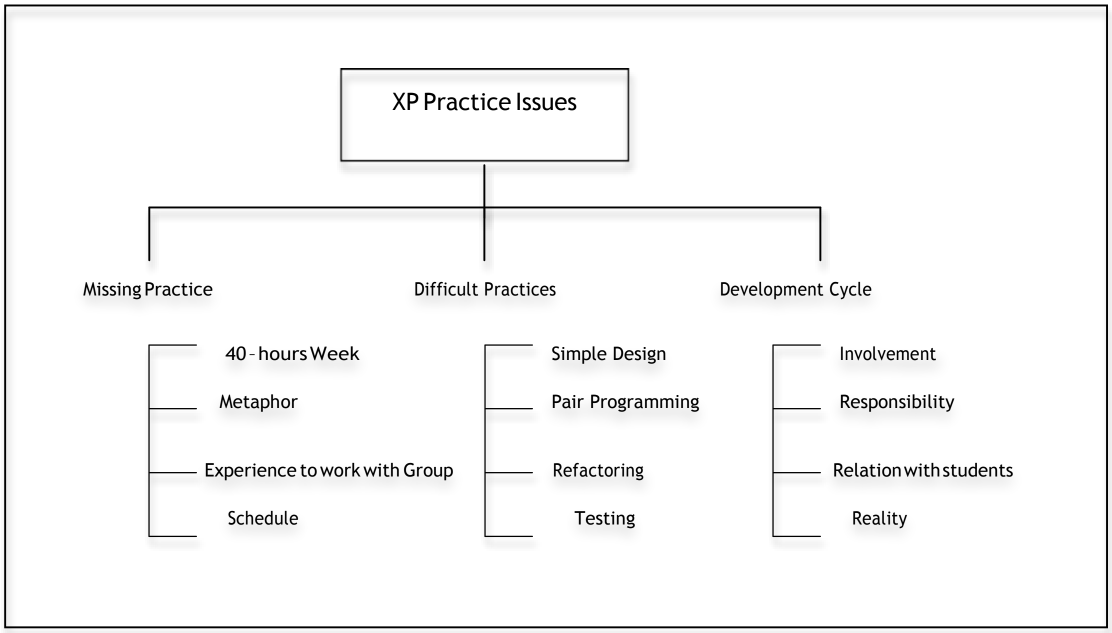

FIGURE (3): XP Practices

6.4 Assigned Project

Any project assigned to the students must take into the consideration:

(1) Complexity: Students will be introduced to a new method (Extreme Programming), so there is no need to make the project very complicated. The main aim is to follow the rules of XP not test their skills in solving the problems.

(2) Project time: the Students have fixed time during the academic semester. It is impossible to request them to work on a class after its due.

(3) Scope: the project definition, requirements, and constraints must clearly be explained to the students. See below figure (4).

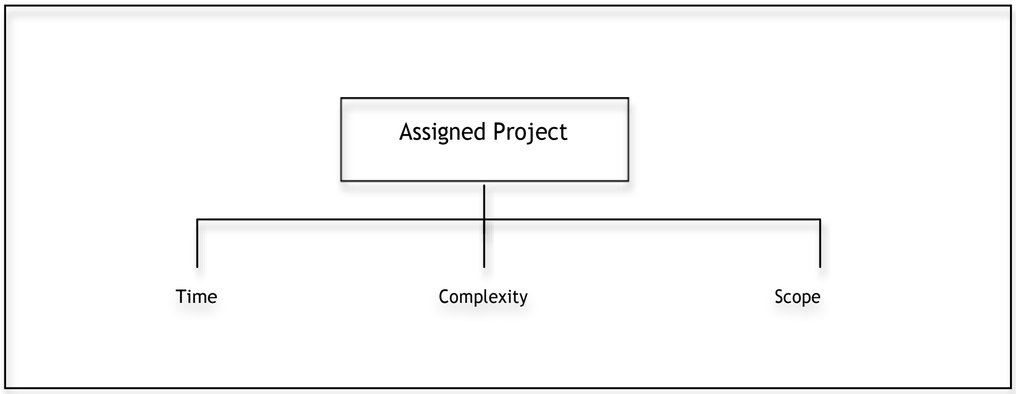

FIGURE (4): Three Important Issues Related to the Assigned Project.

6.5 Communication and Collaboration

Effective communication is core value in agile development. It is the only way to keep the team informative and productive. In the educational environment, the communication is one of the major problems due to the conflict of the students' schedules. Relying students only on their attendances on the regular times for the classes will not be enough to build the relation among the team members. Students have to pay extra time to be spent working on the project outside of the academic regular times. Student's attendance in the lab and other team meetings, have to be taken by team members seriously. As mentioned previously, students will not work for one class 40 hours per week; so using other communication tools such as (Skype, wiki, email, and phone) are recommended. The communications factors are pictured in figure (5).

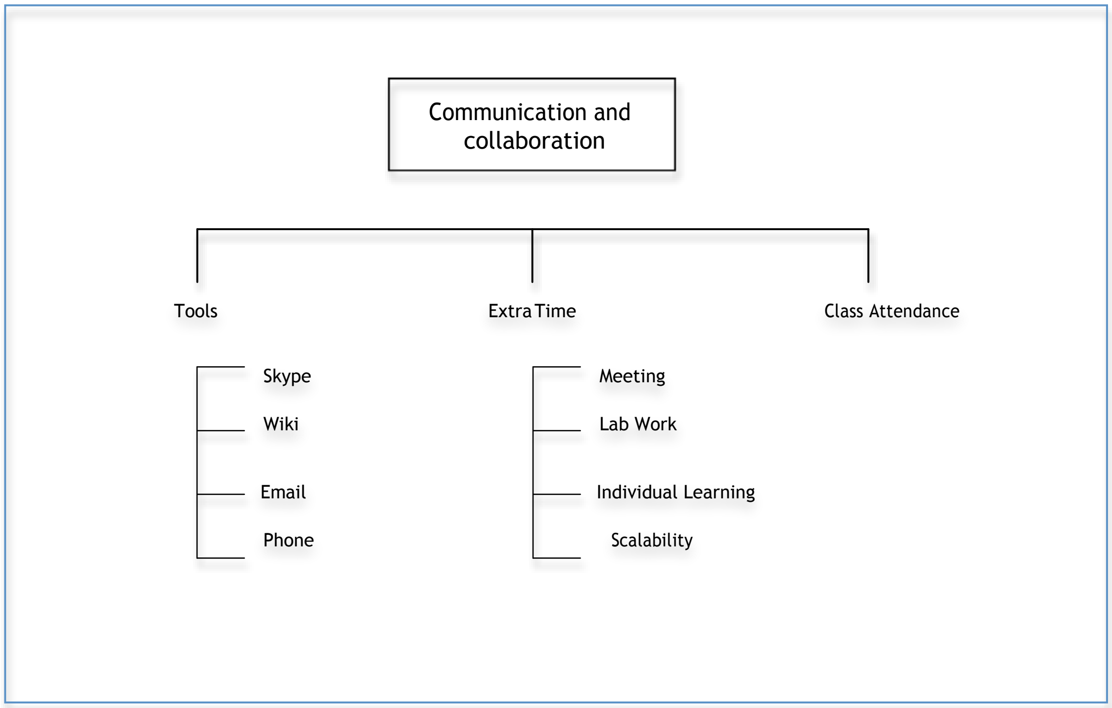

FIGURE (5): Communication and Collaboration

6.6 Environment Sitting

The XP on the industry environment has higher advantages in the workplace sitting more than the labs in the educational environment. However, Students must adjust the labs and some equipment's spot to facilitate face-to-face communication. The convenient workstation is main factor in encouraging students to work together and exchange the information quicker. Also, the social activities are needed to build the relation among the team members. The instructor also should provide the colorful cards for stories and watching the board with card movements. See figure (6).

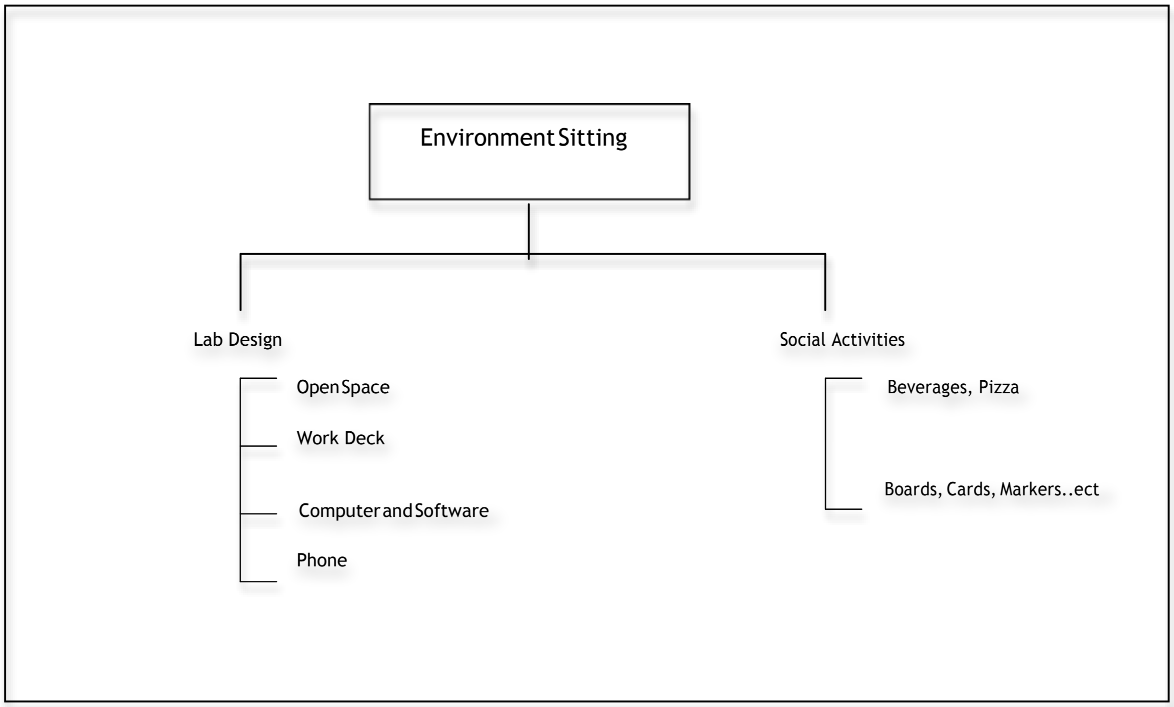

FIGURE (6): Environment Sitting

Figure (7) illustrates the important elements when applying Extreme programming in educational experiment.

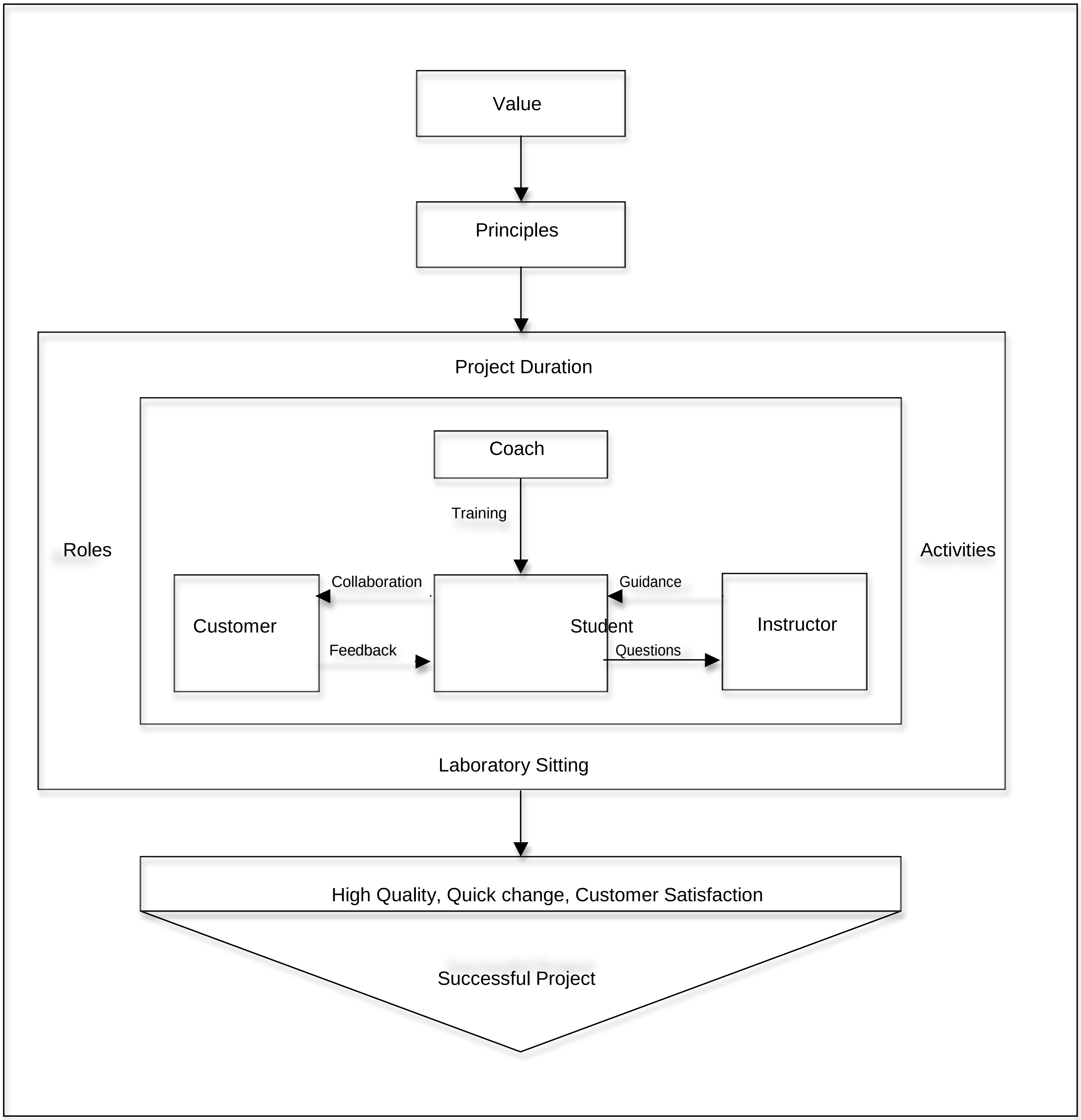

FIGURE (7): Essential Elements in XP Experiment in Education.

7. CONCLUSION

We found the applicability of using the Extreme programming in the educational fields especially when having students as developers. Some of the XP practices were successfully adopted, while the others were not. The students programming skills are fundamental requirements. Defiantly, the students experience will have positive impacts in the whole XP experiment. The pair programming was a new technique to many students; but they found it enjoyable and useful. Here is where the researcher needs to keep in mind before performing the XP experiment in the educational environment.

- One-site customer will help the students to overcome some of the new techniques and principles in a proper way. The students have to be trained in the beginning of the project and other XP materials (papers and books) should be provided for reading.
- Instructor has to devote his time in working with students and provide detailed explanation for their questions. In case of taking the role of the customer, the instructor has to be available all the time and fully involved in the project in each step. The skills of acting two roles (teaching and one-site customer) and scaling multi teams will be essentially needed.
- Simple design, testing, and refactoring require collective work by all the team members including the coach and instructor assistance.
- Communication and relation among the students must to be strongly built to increase the productivity.
-

Extra time for social activates will be beneficial in this matter.

- Workplace and scheduling problems will remain as core problems and difficult to adjust. However, any effort to accommodate the XP project in the educational sitting will be resulted in elevating the performance and quality.
- XP experiment requires an additional person to observe the teams' work and watch all the process to write the notes and comments.

In our feature work, we intend to weight all the XP practices and propose a more detailed description of all the factors that have direct influence on each practice. In addition, we will provide appropriate metrics to measure the student's performance and productivity.

8. REFERENCES

[1] Martin, Robert. Agile Software Development-principles, patterns, and practices. Upper Saddle River, New Jersey, 2003.

[2] Matthias M and Walter F, Extreme Programming in a University Environment, in: Proceedings of the 23rd International Conference on, Page(s):537-544,University Karlsruhe, Germany, May 2001.

[3] Norsaremah Salleh, Azlin Nordin, and Hamwira Yaacob. Experimenting with Pair Programming in a 2nd year Java Programming Course. International Islamic University, Malaysia.

[4] Jean-Guy Schneider and Lorraine Johnston. eXtreme Programming at Universities-An Educational Perspective, in: Proceedings. 25th International Conference on , Swinburne University of Technology, Page(s): 594 – 599, May 2003.

[5] Peter Maher. Weaving Agile Software Development Teaching into a Traditional Computer Science Curriculum, Information Technology: New Generations, 2009. ITNG '09. Sixth International Conference on, Page(s): 1687 – 1688, Webster University, April 2009.

[6] Leurie Williams, Lucas Layman, Jason Osborne, Neha Katira. Examining the Compatibility of Student Pair Programming, Agile Conference, 2006 ,Page(s): 10 pp. – 420, North Caroline Stat University, July 2006.

[7] Carol Wellington, Thomas Briggs, and C. Dudley Girard. The Impact of Agility on a Bachelor’s Degree in Computer Science, Agile Conference, 2006 , Page(s): 5 pp. – 404, July 2006.

[8] Jeremy Kivi, Dalene Haydon, and Jason Hayes. Extreme Programming: A University Team Design Experience, Electrical and Computer Engineering, 2000 Canadian Conference on, Page(s): 816 - 820 vol.2, University of Calgary, 2000.

[9] Carol A. Wellington. Managing a Project Course Using Extreme Programming. Shippensburg, Frontiers in Education, 2005. FIE '05 in: Proceedings 35th Annual Conference, Page(s): T3G – 1, 19-22 Oct, University Shippensburg, 2005.

[10] Gorel Hedin, Lars Bendix, Boris Magnusson. Magnusson, B. Introducing Software Engineering y means of Extreme Programming, in: Proceedings. 25th International Conference on, Page(s): 586 – 593, Lund Institute of Technology, Sweden, May 2003.

[11] Noel F LeJeune. Teaching software engineering practices with Extreme Programming. Comput. Small Coll., Vol. 21, No. 3. (2006), 107-117.

[12] Rick Mugridge, Bruce MacDonald, Partha Roop, Ewan Tempero. Five Challenges in Teaching XP, in: Proceeding XP'03 Proceedings of the 4th international conference on Extreme programming and agile processes in software engineering, University of Auckland, New Zealand, 2003.

[13] Grigori Melnik, Frank Maurer. Perceptions of Agile Practices: A Student Survey, in: Proceedings In Extreme Programming and Agile Methods - XP/Agile Universe, page(s): 241-250, University of Calgary, 2002.

[14] Jennifer Bevan, Linda Werner, Charlie McDowell. Guidelines for the use of pair programming in a freshman programming class, in: Proceedings the 15th Conference on Software Engineering Education and Training, California University, Santa Cruz, CA, 2002.

[15] Fran Keenan. Teaching And Learning XP, in: Proceedings the 3 rd International conference on XP, Dundalk Institute of Technology, 2002.

[16] Mike Holcombe, Marian Gheorghe, Francisco Macias. Teaching XP for Real: some initial observation and plans, University of Sheffield.

[17] Reichlmayr, Thomas. The Agile approach in An Understanding Software Engineering Course, in: Proceedings in the 33rd Annual on Frontiers in Education, Page(s): S2C - 13-18 vol.3, 2003.

[18] David Astels, Granville Miller, Miroslav Novak. A practical Guide to Extreme Programming. Prentice Hall PTR, 2002.

[19] Kent Beck. Extreme Programming Explained Embrance Change. Addison-Wesley,1999.

[20] A Taxonomy for "Bad Code Smells": < http://www.soberit.hut.fi/mmantyla/badcodesmellstaxonomy.htm> (accessed 3.4.11).

[21] Jennifer Bevan, Linda Werner, Charlie McDowell. Guidelines for the use of pair programming in a freshman programming class, in: Proceedings the 15th Conference on Software Engineering Education and Training, California University, Santa Cruz, CA, 2002.

[22] Anuja Shukla, Laurie Williams, Adapting extreme programming for a core software engineering course, in: Proceedings the 15th Conference on Software Engineering Education and Training, Page(s): 184 – 191,2002.

[23] Ken Schwaber, Mike Beedle, Agile Software Development with SCRUM. Prentice Hall, 2001.

[24] NooP.LN:<http://www.noop.nl/2009/04/the-optimal-team-size-is-five.html> (accessed 22.4.11).

[25] Jeff Hodgkinson. Communications Is the Key to Project Success,2009.